hochschule mannheim

Techniken der Programmentwicklung (TPE)
Sommersemester 2017

FAKULTÄT FÜR INFORMATIK

Pflichtübung 1

Ausgabe:	16.03.2017
Abgabe:	29.03.2017 (23:50 Uhr)
Testat:	30.03.2017 (9:45–12:15 Uhr)

Aufgabe 1: Pakete — 10 Punkte

(a) Wählen Sie ein Paket-Benennungs-Schema für Ihre Übungsaufgaben zur Vorlesung TPE. Erklären Sie Ihr Schema und begründen Sie, warum es durch das von Ihnen gewählte Schema nicht zu Kollisionen mit anderen Gruppen und Entwicklern kommen kann.

Abgabe – Schriftliche Begründung der Namenswahl.

Aufgabe 2: Devisenkonto — 90 Punkte

Um Ihr Studium zu finanzieren, haben Sie sich eine Anstellung bei der *International Lies and Deception Bank Corporation* gesucht. Diese Bank zeichnet sich durch erstklassige Beziehungen nach Nord-Korea und ihr internationales Engagement aus. Kurz nach Ihrem ersten Arbeitstag kommen Ihnen Gerüchte zu Ohren, dass der Vorstandsvorsitzende Louis Cyphre sehr ruppig mit Mitarbeitern umgeht, die schlechte oder fehlerhafte Software entwickeln. Besonderen Wert legt er auf sauberen Programmierstil und gute Tests. Hierdurch mehr als motiviert, stürzen Sie sich sofort in die Arbeit. Ihre erste Aufgabe ist die Implementierung einer Software zur Kontenverwaltung für die Bank.

Da die Bank international tätig ist, muss zu jedem Betrag bekannt sein, in welcher Währung er vorliegt. Beträge (einschließlich der dazugehörigen Währung) werden von einer speziellen Klasse `Betrag` verwaltet. Die Daten zu einer Währung (z. B. Wechselkurs) liegen in der Klasse `Waehrung`. Ein Konto bei der Bank wird durch die Klasse `Konto` repräsentiert.

Im Bereich finanzmathematischer Berechnungen setzt man normalerweise keine Fließkommazahlen ein, da diese durch Rundungsfehler zu ungenauen Ergebnissen führen. Verwenden Sie daher bei allen folgende Klassen für die Darstellung von Geldbeträgen grundsätzlich den Datentyp `long` und speichern Sie die Daten mit einer Genauigkeit von zwei Nachkommastellen, d. h. dass zum Beispiel ein Betrag von `19,45` als `1945` gespeichert wird.

(a) **Verwaltung von Währungen**

Schreiben Sie eine Klasse `Waehrung`, die folgende Informationen zu einer Währung verwalten kann:

- *Name* der Währung (`name`), z. B. Euro oder Rubel
- *Kürzel* der Währung (`kuerzel`), z. B. € oder RUB
- *Wechselkurs zum Dollar* (`kurs`)

Die Daten sollen über den Konstruktor in das Objekt gelangen und danach nicht mehr verändert werden können, d. h. die Klasse soll *immutable* sein. Über entsprechende Methoden (`getKuerzel()`, `getKurs()`, `getName()`) können die Informationen wieder ausgelesen werden.

Weiterhin soll es eine Methode `umrechnen(...)` geben, mit der man Beträge von einer Währung in eine andere umrechnen kann. Hierzu wird eine Betrag als `long` und eine Zielwährung übergeben und die Methode gibt den umgerechneten Betrag wieder als `long` zurück. Um nicht den Wechselkurs zwischen allen Währungen speichern zu müssen, werden verschiedene Währungen immer über den Dollar umgerechnet. Sie rechnen also zuerst den Betrag in Dollar um und danach in die Zielwährung.

Weiterhin besitzt die Klasse noch drei weitere Methoden:

- `toString()` – gibt die Informationen zur Währung als String zurück. Die Ausgabe soll den Namen der Währung, das Kürzel und den Wechselkurs (mit vier Nachkommastellen) enthalten. Eine beispielhafte Ausgabe sähe wie folgt aus: `Rubel [RUB] 1 $ = 0,0254 RUB`
- `equals(Object o)` – vergleicht den Inhalt des Objekts mit einem anderen. Sie können sich die Methode von Eclipse generieren lassen.
- `hashCode()` – berechnet einen Hash-Wert über das Objekt. Sie können sich die Methode von Eclipse generieren lassen.

Abgabe – Quelltext der Klasse `Waehrung`.

(b) **Liste von Währungen**

Schreiben Sie eine Klasse `Waehrungen`, die vorgefertigte Objekte von Währungen mit den jeweiligen Wechselkursen enthält, damit man diese einfach innerhalb des Programms wiederverwenden kann.

Folgende Kurse sind vorgegeben:

Name	Kürzel	Kurs zum $
US-Dollar	$	1,0000
Euro	€	1,2690
Yen	¥	0,0091
Rubel	RUB	0,0255
Schweizer Franken	CHF	1,0509

Abgabe – Quelltext der Klasse `Waehrungen`.

(c) **Verwaltung von Beträgen**

Die Beträge werden durch die Klasse `Betrag` realisiert.

Die Klasse hat folgende Eigenschaften:

- Die Objekte der Klasse sind *unveränderlich (immutable)*, d. h. die Daten eines Objekts können nach der Erzeugung nicht mehr verändert werden.
- Beim Anlegen eines Objektes muss man sowohl den Betrag als auch die Währung angeben. Man kann den Betrag entweder als `double` mit Kommastellen angeben oder als `long`, wobei dann die Angabe in $\frac{1}{100}$el erfolgt. Der Betrag wird (wie bereits erwähnt) intern immer als `long` gespeichert. Wird er als `double` angegeben, werden nur maximal zwei Nachkommastellen berücksichtigt (danach wird abgeschnitten).
- Die Klasse kann sowohl positive als auch negative Beträge verwalten. Um festzustellen, ob der Betrag negativ ist, kann man die Methode `getVorzeichen()` aufrufen, die entweder `1` für positiv oder `-1` für negative Werte zurückgibt.
- Die Klasse bietet elementare arithmetische Operationen an, die nun beschrieben werden.

Implementieren Sie bitte die folgenden Methoden in der Klasse `Betrag` (Methodennamen in Klammern):

- Addieren zweier Beträge (`addiere(...)`)
- Subtrahieren zweier Beträge (`subtrahiere(...)`)
- Multiplizieren eines Betrags mit einem `double`-Wert (`multipliziere(...)`)
- Multiplizieren eines Betrags mit einem `int`-Wert (`multipliziere(...)`)
- Berechnen eines Prozentwertes von einem Betrag (`prozent(...)`)
- Berechnen eines Promillewertes von einem Betrag (`promille(...)`)
- Auslesen des Wertes vor dem Komma (`getVorkomma()`). Das Vorzeichen wird hier *nicht* berücksichtigt.
- Auslesen des Wertes nach dem Komma. Hier können nur Zahlen zwischen 0 und 99 zurück gegeben werden. (`getNachkomma()`)
- Auslesen des gesamten Wertes als String mit Währungsangabe und führender Null bei Werten kleiner als 1. Das Vorzeichen wird bei negativen Beträgen ebenfalls mit ausgegeben. Zum Beispiel `0,99 €`, `12,90 €`, `11,00 €`, `-0,09 RUB` oder `0,00 CHF`. (`toString()`)
- Konvertieren der Zahl in ein `double`-Wert, wobei aber nur zwei Nachkommastellen im zurück gegebenen Wert vorhanden sein dürfen (`getAsDouble()`)
- Vergleich zweier Beträge auf Gleichheit, wobei zwei Zahlen nur gleich sind, wenn sie in Betrag und Währung übereinstimmen (`equals(...)`). Sie können sich diese Methode von Eclipse generieren lassen.

Abgabe – Quelltext der Klasse `Betrag`.

(d) **Konto**

Implementieren Sie eine Klasse `Konto`, die Buchungen verwalten kann. Jedes Konto hat einen Inhaber (`inhaber`, `getInhaber()`) und eine Währung, auf die das Konto lautet (`waehrung`, `getWaehrung()`).

Über die Methode `buche(...)` können Beträge auf das Konto gebucht werden (positive wie negative). Wenn der gebuchte Betrag in einer anderen Fremdwährung erfolgt, d. h. in einer Währung, die von der Kontowährung abweicht, wird der Betrag automatisch zum jeweiligen Wechselkurs in die Kontowährung umgerechnet.

Mit der Methode `saldo()` kann man den Saldo auf dem Konto abfragen, d. h. die Summe aller Buchungen.

Über die Methode `gebuehren(...)` kann die Bank automatisch einen gewissen Promillesatz an Gebühren vom Konto abziehen. Damit ergibt sich die Gebühr als das Produkt aus Saldo und Promillesatz.

Die Methode `toString()` erlaubt es, einen Auszug des Kontos zu bekommen. Dieser soll sich an folgendem Beispiel orientieren:

```
Kontoinhaber: Peter Lustig
Währung: US-Dollar
---------
10,00 $
20,00 $
33,33 $
-10,00 $
-0,79 $
---------
Saldo: 52,54 $
```

Abgabe – Quelltext der Klasse `Konto`.

(e) **Tests** Überprüfen Sie die Funktionalität Ihrer Implementierung mit entsprechenden JUnit-Tests und weisen Sie mit diesen Tests nach, dass die implementierten Operationen richtig funktionieren.

Abgabe – Quelltext der JUnit-Tests.

Achtung
Bitte beachten Sie folgendes, damit es nicht zu unnötigen Punktabzügen in der Bewertung kommt:

- Benennen Sie Klassen und Methoden konsistent und verständlich. Klassen sollten Nomen als Namen, Methoden Verben haben.
- Dokumentieren Sie alle Klassen, Interfaces, Konstruktoren und Methoden mit JavaDoc. Dokumentieren Sie auch alle Parameter, Rückgabewerte und Ausnahmen.
- Formatieren Sie Ihren Code konsistent. Ein guter Standard sind 4 Spaces Einrückung pro Ebene ohne Verwendung von Tabulatoren.
- Halten Sie sich an die Sun Java-Code-Convention
  http://www.oracle.com/technetwork/java/codeconventions-150003.pdf
- Programmieren Sie defensiv und testen Sie Eingabewerte auf deren Gültigkeit. Ihr Programm darf auch mit Daten nicht abstürzen, die außerhalb der Aufgabenstellung liegen.
- Machen Sie keine Konsoleneingaben oder -Ausgaben, es sei denn die Aufgabe fordert dies explizit.
- Halten Sie Daten und Methoden zusammen. Trennen Sie diese nicht unnötig auf.
- Kopieren Sie keinen Code sondern versuchen Sie mit den bekannten Mitteln der Objektorientierung Code-Duplikate zu vermeiden.